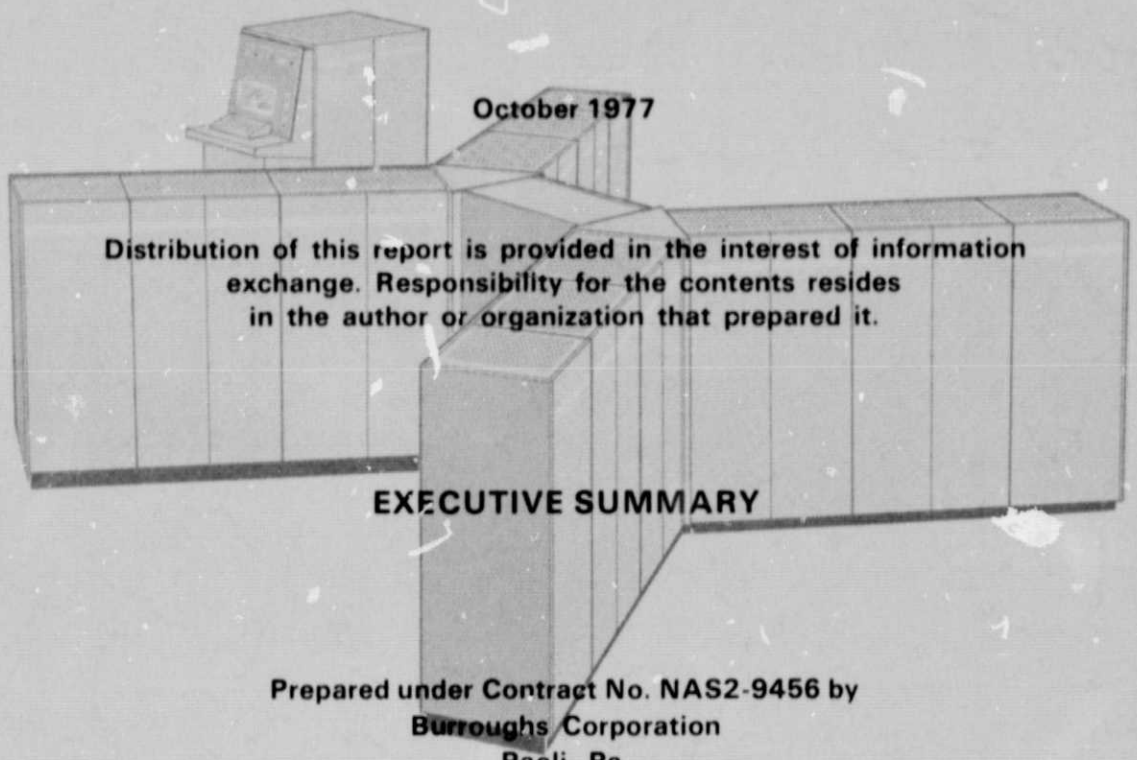
CR-152060

UNCLASSIFIED

FINAL REPORT

NUMERICAL AERODYNAMIC SIMULATION FACILITY

PRELIMINARY STUDY



October 1977

EXECUTIVE SUMMARY

Prepared under Contract No. NAS2-9456 by
Burroughs Corporation
Paoli, Pa.

for

AMES RESEARCH CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

UNCLASSIFIED

FINAL REPORT

NUMERICAL AERODYNAMIC SIMULATION FACILITY

PRELIMINARY STUDY

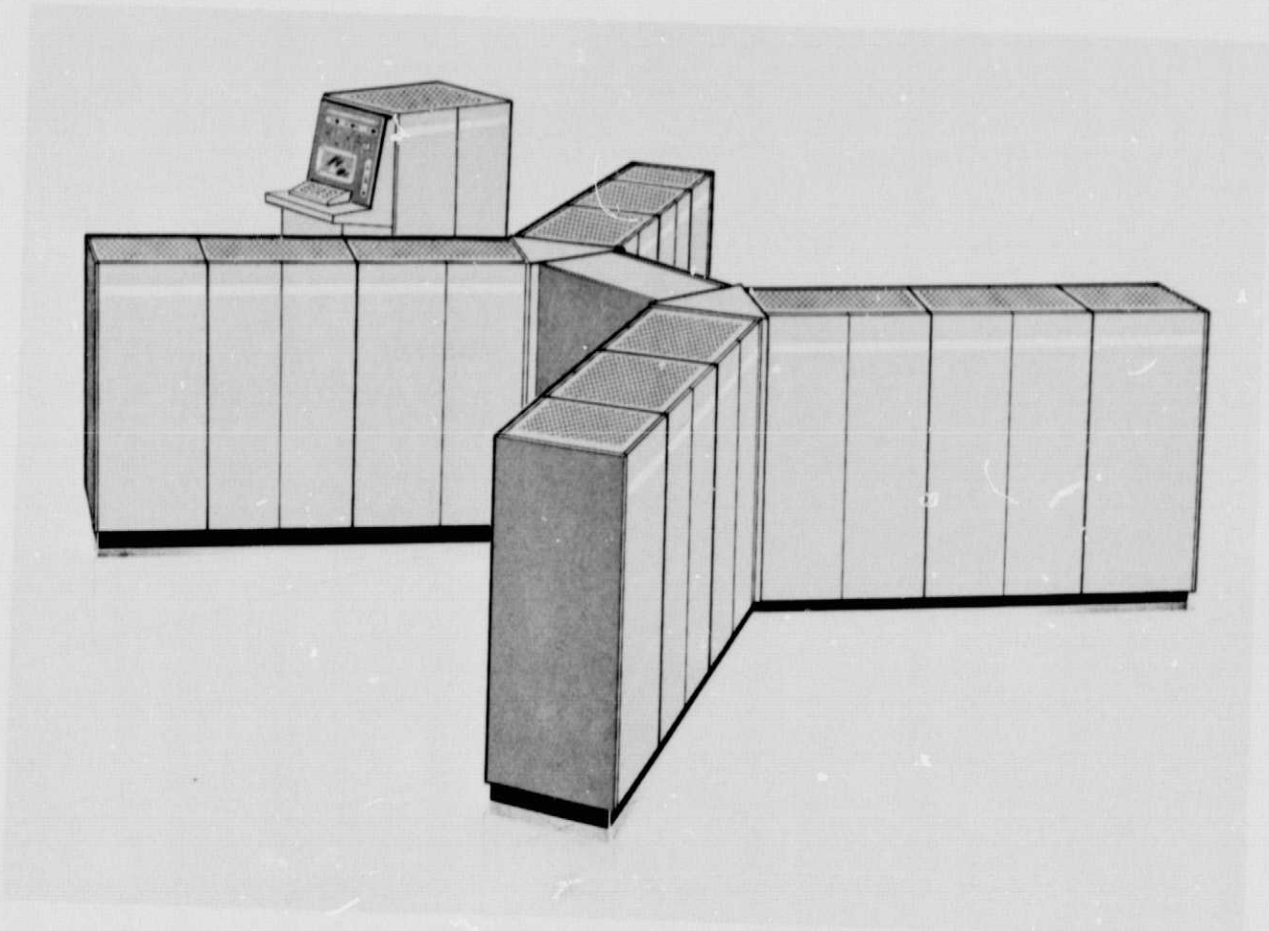
October 1977

EXECUTIVE SUMMARY

Prepared under Contract No. NAS2-9456 by
Burroughs Corporation
Paoli, Pa.

for

AMES RESEARCH CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION



NAVIER-STOKES SOLVER

INTRODUCTION

Burroughs Corporation is pleased to submit this executive summary of the findings of the Numerical Aerodynamic Simulation Facility (NASF) Preliminary Study. This report presents a unique solution to the problem of numeric aerodynamic simulation. The solution consists of a computing system designed to meet the stated objective of providing an effective throughput of one billion floating point operations per second for three-dimensional Navier-Stokes codes. Burroughs presents this design with full confidence that it is feasible to complete the detailed design and construction of this machine within the required time-frame. This high level of confidence is based on Burroughs' extensive and continuing experience in the design and development of very high performance computer systems. It is Burroughs' belief that the computer industry will not produce a commercial general purpose machine with the required performance by the early 1980's. Consequently, we feel that the design and construction of a relatively specialized system is not only feasible, but necessary to the achievement of NASF objectives.

This view is based on two business judgements. First, projections of both computing-power and cost of performance of commercial computers for the 1980 to 1985 time-frame do not include a machine of this capacity or price. That is, a generation gap will exist between any NASF implementation and concurrent commercial products. Second, market trends indicate that an insufficient market exists to sustain development of a machine with two orders of magnitude speed increase on a commercial basis.

In summary, we believe that the system presented in this report constitutes the best approach to meeting the NASF goals in a timely and cost-effective manner, and that NASA has an opportunity to maintain a "forefront" position in the scientific community while achieving these goals.

The results of this study have produced a unique solution to the problem of numerical aerodynamic simulation of three-dimensional Navier-Stokes equations. In order to fully appreciate the design, its features, and subtleties, the methodology of the study which evolved this solution must be understood. This executive summary is intended to explain that methodology. First, the problem and solution, in brief, will be presented, then basics of the study approach will be explained. Next, a description of each of three substudies follows with emphasis on specifically what was examined and why. Finally, the results of the substudies are merged to highlight their impact on the processor architecture evolution, and show how the "baseline design" for NASF was selected. The final report chapters will discuss details of that design.

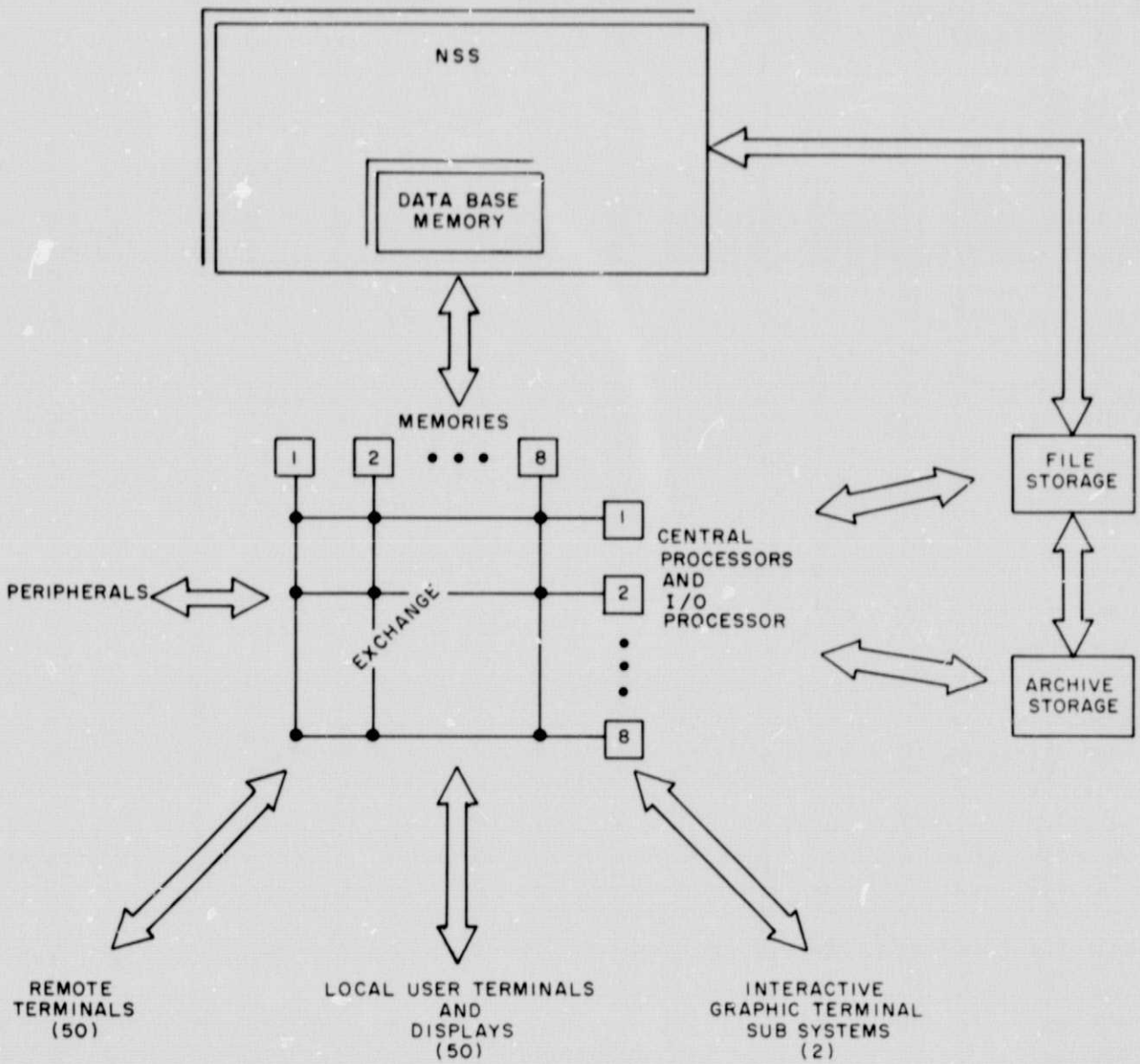


Figure 1 NASF System Block Diagram

STUDY OBJECTIVES

The Numerical Aerodynamic Simulation Facility Preliminary Study Objectives were to determine the feasibility of designing a system delivering one billion floating point operations per second effective throughput for three-dimensional Navier-Stokes codes by 1982. If feasible, a processor architecture and functional design definition were to be developed, supporting that assertion, with attendant requirements of power, size, cost, schedule, etc.

NASF OVERVIEW

The basic structure of the candidate baseline NASF system is shown in Fig. 1. The major elements are:

- The Host, a Burroughs B7800 multiprocessing system
- The Navier-Stokes Solver (NSS) . . . the high throughput work-horse of the system
- File Memory
- An Archival Storage system.

THE HOST COMPUTER

The Host, a Burroughs B7800 system, acts as the system manager and support facility. It provides the user interface, schedules and dispatches NSS tasks, and executes supporting functions such as compilation, data reduction, and output generation.

THE NAVIER-STOKES SOLVER (NSS)

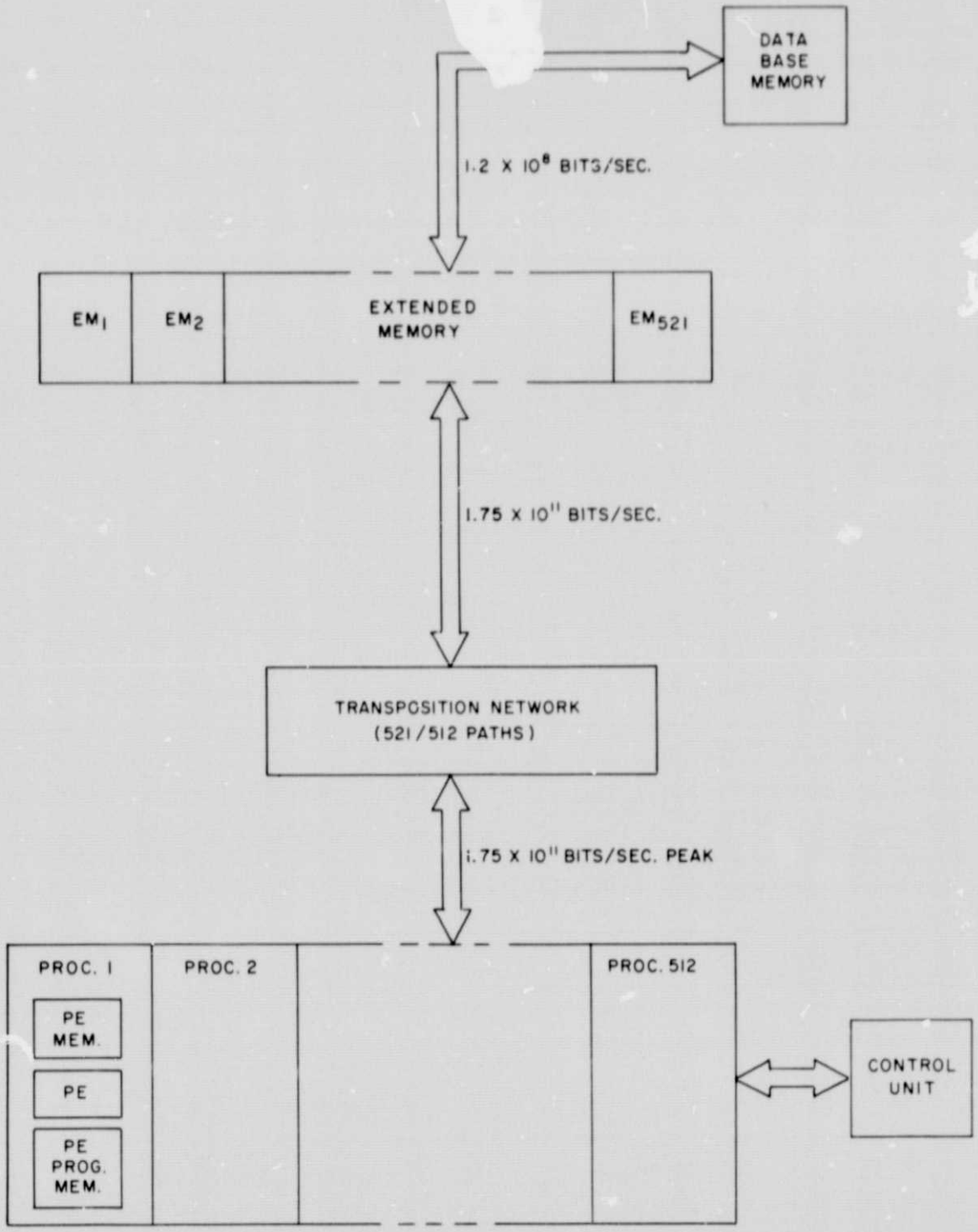
The NSS is the high throughput computational element. It is a highly parallel processing array, designed to provide the required computational throughput on three-dimensional Navier-Stokes programs. The Data Base Memory (DBM) of the NSS provides the interface between the NSS and other system elements. The program and data files are loaded to the DBM by the Host. The NSS with the DBM constitute a high speed "computational envelope," allowing the NSS to run at maximum speed essentially without outside interruption or dependence until job completion.

THE ARCHIVE MEMORY

The Archive provides a very large storage capability for long term retention of programs and data bases. It consists of a commercially available mass memory system, which is managed by the Host.

THE FILE MEMORY (FM)

The FM provides for short term file retention, staging and buffering between the Host, the Archive, and the DBM. It consists of a standard disk pack sub-system, and is also managed by the Host.


DATA BASE MEMORY
1.2 X 10^8 BITS/SEC.
EM1
EM2
EXTENDED MEMORY
EM521
1.75 X 10^11 BITS/SEC.
TRANSPOSITION NETWORK (521/512 PATHS)
1.75 X 10^11 BITS/SEC. PEAK
PROC. 1
PE MEM.
PE
PE PROG. MEM.
PROC. 2
PROC. 512
CONTROL UNIT

MAJOR ELEMENTS OF THE NAVIER-STOKES SOLVER

The principal innovation in the NASF system is the NSS. The organization of the NSS is shown in Figure 2, and its characteristics are summarized in Table 1. The major features of this processing array are:

- Highly Parallel Architecture
  The NSS consists of 512 computational processors, each with its own local data program memories. These are coordinated by a single control unit, and connected via a transposition network to 521 modules of extended memory.

- Synchronizable Operation
  This feature of the NSS suggests the name we have given to the computational array, the Synchronizable Array Machine, or SAM. Previous processor arrays have operated in "LOCKSTEP," essentially synchronizing on every instruction cycle. The computational array of the NSS is synchronized explicitly by the code stream only when necessary. Between synchronization points, the individual processing elements may operate asynchronously, allowing them a degree of freedom in scheduling instruction sequences.

- Conflict Free Memory Access
  The transposition network between the processing elements and extended memory allows conflict free access to vectors in any dimension at full memory bandwidth. This eliminates the non-productive time which would otherwise be consumed by reordering or transposition of data before processing.

- Large Second Level Store
  The Data Base Memory (DBM) in the NSS provides an interface between NSS and Host that allows each to process independently of the other. NSS processing need never be held up waiting for some response from the Host.

- System Balance
  All transfer rates and execution speeds are tuned to one another in concert with the requirements of the application. This provides for high efficiency by balancing the utilization of system elements.

- Ease of Use
  A high level user language, complemented by an instruction set oriented to efficient implementation of high level language programs, allows ready access to the computational power of the NSS, without encumbering the user with assembly language programming or implementation details.

TABLE 1 NSS CHARACTERISTICS

Computational Capacity (On instruction mix)	1.7×10^9 floating operations/sec.		
Number of Processing Elements Number of Extended Memory Modules	512 521		
Memory capacities (total) Extended memory Processing element memories Processing element program memories	 34 million words 8 million words 4 million words		
Transfer rates (bits/sec)	per path	no. paths	total
PE - PEM	490×10^6	512	2.5×10^{11}
PE - PEPM	490×10^6	512	2.5×10^{11}
PE - (PEM+PEPM)	10^9	512	5×10^{11}
EM - via TN - PEM streaming mode 1 word/transfer	 4×10^8 1×10^8	 512 512	 2×10^{11} 5.5×10^{10}
EM - DBM	–	–	1.4×10^8
Program loading to all PE's simultaneously			4×10^8 per PE
Clock, synchronous throughout the NSS	50 MHz minor cycles 25 MHz major cycles		
Total No. of IC packages, including memory (almost all LSI)	200,000		
Word Size:	48 Bits		

STUDY METHODOLOGY

Experience in the design and manufacture of data processing equipment, especially very-high-performance computer systems, leaves many lessons behind. In addition to knowing what a design team should do, there are some lessons about what should **not** be done.

The Burroughs study team took care to avoid a serious problem that often traps those aiming at maximum speed – namely pushing the state of the art on too many frontiers. One could rely on significant advances in

- Architecture,
- Hardware Technology, or
- Software Technology.

For increased performance Burroughs chose Advanced Architecture taking care to build on mature or developed software whenever possible. In addition hardware implementation will be conservative, consistent with performance goals, and will not rely on imposing inordinate speed requirements or new, untried technologies.

Selecting architectural elegance as the new frontier, the study concentrated on matching the architecture to the problem. Existing computer structures were not integrated to force-fit a "super-structure" of these units to the problem. The reasons were:

- Lack of Architectural Flexibility
- Inefficient and Not-Cost-Effective.

Although performance requirements may be met in this fashion, the lack of architectural freedom with the structures implies that many hardware and software elements are not utilized, others must be customized, resulting in a machine that has some "dead-wood."

The NASF system presented here was developed by evolution from careful analysis of the problem characteristics to insure a genuine fit. Top-down design fundamentals were practiced so that on each of the several design iterations, results could be traced to assumptions. Traceability of this sort allows bottlenecks or errors found to be identified at their origin where viable alternatives could be reexamined.

SUB-STUDIES

Specifically, three sub-studies were executed simultaneously as required by the original contract statement of work.

- THE TECHNOLOGY STUDY developed a data base of logic and memory technologies by literature searches, vendor interviews and conferences, etc. Trends of critical issues and parameters of these technologies were studied and a technology forecast developed for the 1980-1985 time-frame.

- THE MATCHING STUDY analysed the flow models and their characteristics and matched them against candidate processor architectures.

- THE FACILITY STUDY established metrics for the total facility and, at a more detailed level, the facility issues addressing the "buildability" of the final system.

Each sub-study was executed with two objective as shown in Figure 3.

- How do results affect processor architecture choice?
- How do results affect specific design choices in the baseline design?

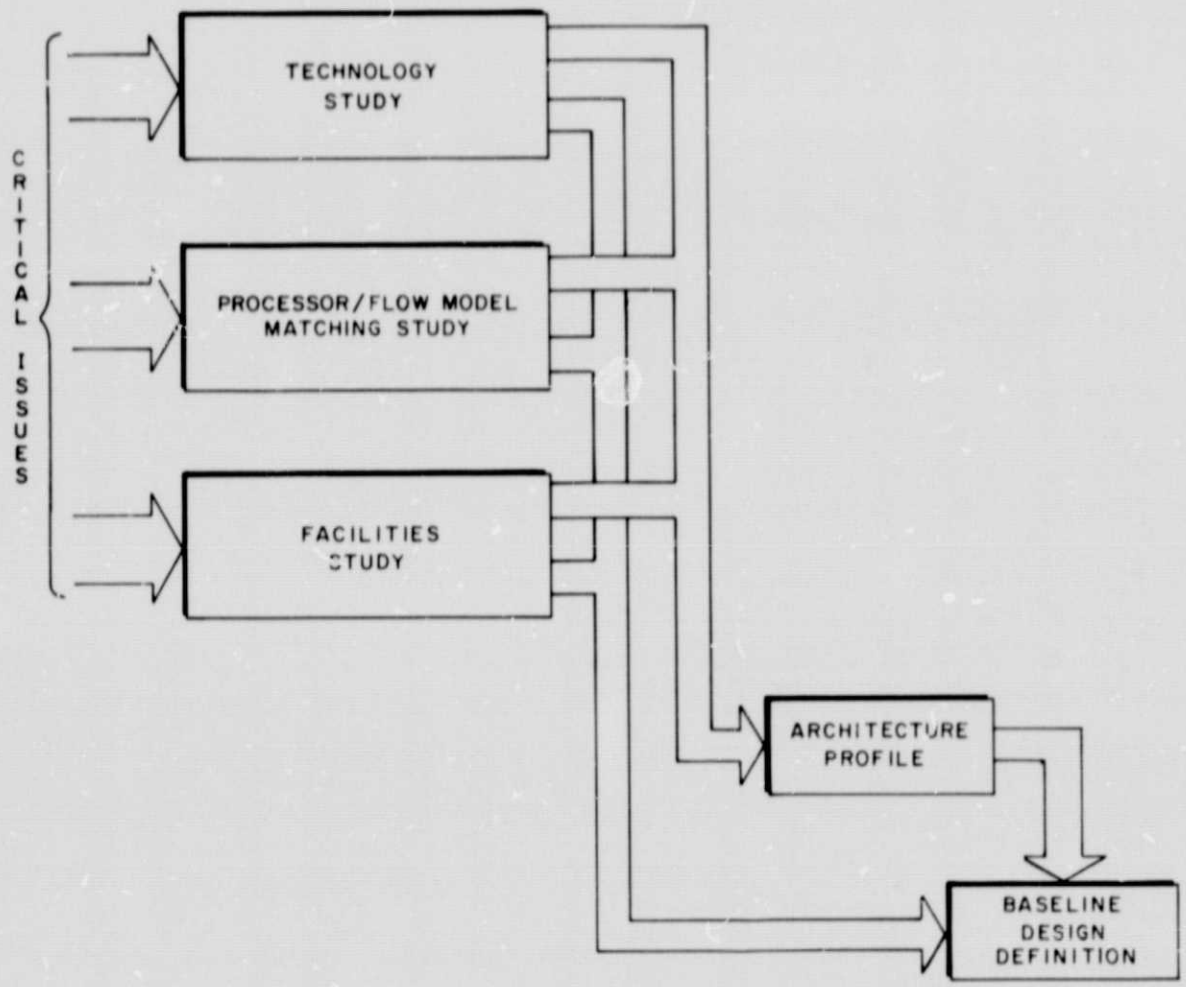


Figure 3 NASF Study Approach

That is, first a processor architecture was evolved as a result of the sub-studies, then a second iteration of the studies supported a more detailed design to the functional design level referred to as the Baseline Design. The result is an NASF definition that directly addresses the salient issues of the problem itself. This NASF definition meets or exceeds all requirements and can be built with a high degree of confidence - an assertion of great significance for such an ambitious task.

TECHNOLOGY STUDY OVERVIEW

The objective of this phase of the study was to establish a technology forecast for the NASF time-frame and assess which logic and memory technologies are most appropriate for the design of such a facility.

The approach taken consisted of the following four tasks:

- Data Gathering
- Establish Critical Issues
- Examine Technologies & Trends
- Extrapolate 1980-85 Forecast.

Data gathering consisted of a three phase effort: a comprehensive literature search, trade conferences and workshops, and interviews with vendors and suppliers such as Motorola, Fairchild, National Semiconductor, Intel, Signetics, and Texas Instruments.

The critical issues which were established were of two types - those affecting performance and those affecting development.

PERFORMANCE

- speed
- density
- reliability
- power

DEVELOPMENT

- cost
- maturity
- extensiveness
- availability

Metrics for judgement of these issues and clarifications of their importance were then developed and used as criteria in the architecture/design process.

Under performance issues, speed of a logic family may be judged by propagation delay times, while with memory the key figures are read/write times. Density refers to the average number of gates or memory cells per chip. Reliability is largely a function of density since failures frequently occur at the substrate to pin connection, and as the number of pin connections decreases per given function, the reliability increases. Power consumption is a measure of the energy costs and reliability associated with a device. A smaller speed-power product indicates better system performance per kilowatt.

As to developmental issues, cost should be considered in the light of performance per dollar, as well as absolute cost. Maturity is determined by field verification of manufacturer's specification. Another

consideration in selecting a technology is the availability of the devices. In addition, multiple sources for all componentry are essential. These factors are important considerations in the selection of a technology family.

The technology survey provided inputs to the study not only in the obvious area of surveying the implementation of digital logic, but also in some areas of packaging, random access and serial memories, and archives.

From the many technologies used to implement digital logic, three are of sufficient interest to report here:

- **ECL** has been the technology of choice in implementing high-speed digital computers for over ten years. The speed-power product, and hence the amount of processing that can be done per watt of power, has been continually improved, and in the last year some LSI has been available in ECL. ECL is a mature but still developing technology, exemplified by Fairchild's "100K" ECL family. This family could be used as a starting point for a baseline design.

- **I^2L** has much better speed-power product than ECL, allowing far more functions per watt. It is currently too slow for the NASF requirement but both speed and availability of standard parts are improving each year. I^2L would consume considerably less power than ECL and is currently utilized internally in LSI chips where the speed is tolerable.

- **MESFETs** promise another improvement, by an order of magnitude, in the speed-power product as compared to I^2L. They are also very fast; however, they are still in early development. Years of development will be required before the MESFET's technology becomes mature.

From this study we conclude that ECL is the most feasible current technology for implementation of an NASF design, and the base line design will begin with ECL as a starting point.

Memory technology represents an area of low risk for the (NSS). 16K-bit dynamic RAM's (Random Access Memory) are currently available. 16K-bit static RAMS and 64K-bit dynamic RAMS are on the drawing board.

CCD shift register memory is currently available in pilot quantities in the 64K-bits size. Another factor of four in storage size (256K-bits) is expected by 1980.

Manufacturers reported the occurrence of spontaneous errors in CCD memories. This leads to a requirement for continuously monitoring the contents of a CCD memory and rewriting it correctly when bit errors occur.

Present bubble memories put severe complexities into the controlling and driving circuitry, making them very difficult to use.

Sufficient information about the magnetic storages available for the archive was obtained to indicate that there are several commercially available contenders for the archive storage. No effort was made to determine which of today's contenders were likely to be withdrawn from the market in the next two years, nor to uncover the new contenders which are undoubtedly under development.

PROCESSOR - FLOW MODEL MATCHING STUDY OVERVIEW

The key sub-study in this effort was the Matching Study. Certainly, it had the most profound effect on the evolution of SAM as the chosen processor architecture as well as some design details. This sub-study was broken into several tasks prior to the actual matching or evolving process itself.

- Cataloging and examination of pertinent generic architectures for consideration to be used as a starting point.
- Establishment and discussion with NASA-Ames of critical issues and basic requirements and capabilities imposed on the architecture by the problem definition.
- Research and discussion of the fundamental characteristics of the flow models which affect the processor architecture.

Following these tasks, the results were merged with those of the other two sub-studies the total implications of which determined the final architecture.

Generic Architectures considered as starting points were:

- Hybrid system composed of analog computation devices with digital control and storage
- Parallel array architectures with replicated arithmetic units executing the same program on different data achieving performance as a multiple of the number of arithmetic units.
  - Type 1 - Lock-Step synchronous arrays with clock-by-clock tight coupling of arithmetic units
  - Type 2 - Non Lock-Step arrays with coupling at predetermined synchronization points rather than every clock
- Pipeline architectures where operations are streamed through different stages with performance as a multiple of the number of states.

A complete discussion of these generic architectures is found in Appendix L of the final report.

Critical issues, basic requirements and capabilities were jointly developed between the study team and NASA Ames personnel. Topics examined were:

- Navier-Stokes Solver Capabilities
- Programming
- NSS - I/O

NSS CAPABILITIES

The ability to solve the three-dimensional Reynolds averaged Navier-Stokes equations, using both explicit/implicit and totally implicit, dimensionally-split, finite-difference methods.

The ability to compute, at high efficiency, problems containing a variety of boundary conditions which include the independent variables, their derivatives, and other auxilliary variables, a variety of internal and external geometries and a variety of turbulence models ranging from algebraic to seven differential equation descriptions.

The ability to compute solutions for up to one million grid points. This implies a data base range to 14 million words for:

5 conservation variables at 2 time levels

1 turbulence variable

3 grid coordinates

to 40 million words for:

5 conservation variables at 2 time levels.
7 turbulence variables at 2 time levels.
3 grid coordinates
12 metrics (including time)
1 Jacobian

The ability to obtain steady state solutions for one million grid points in 10 minutes of CPU time for 3-D problems using algebraic turbulence models. At present this must be measured using 2-D explicit/implicit and implicit codes as performance metrics.

Two examples of typical programs and their computational requirements are given below:
Explicit code (MacCormack) status: A 2-D airfoil steady-state solution was obtained in 7 minutes on CDC 7600 for 2100 grid points. The steady-state was reached after 13 chord lengths of travel by computing inviscid solution for 7 chords and viscous solution for remaining 6 chords. Effective computing speed on 7600 is about 2 MFLOPS. Assuming twice the computational effort at each grid point for the 3-D case, this implies that to compute 13 chords in 10 minutes for one million grid points requires an effective computing speed of 1.4 gigaflops. Greater efficiencies by 1980 can be expected.

Implicit (Lomax, Steger) code status: A 2-D airfoil steady-state (12 chords traveled) was obtained in 10 minutes on CDC 7600 for 2300 grid points - all calculations were viscous. The effective computing speed on 7600 is about 2 megaflops. This code implies that an effective computing speed of 2 gigaflops will be needed for a 3-D calculation over one million grid points. However, researchers working on the implicit code are confident that improvements in the treatment of boundary conditions and other strategies can improve the speed of the method by a factor of 2 which implies that at least one-gigaflop effective rate will be needed.

It is concluded that the minimum effective computing rate needed for the Navier-Stokes problem is one gigaflop.

A precision of 10 decimal digits is required.

PROGRAMMING

A high level programming language consistent with ease of mapping the solution methods onto the machine, optimum machine performance and the available language development time is necessary.

Desirable programmability features of the Navier-Stokes machine are as follows:
A FORTRAN-like high level language with extensions necessary for efficient problem mapping. As well as the following features.

- a stable optimizing compiler
- good compiler diagnostics
- warning from the compiler of possible run-time inefficiencies
- ability to give good run-time diagnostics and statistics
- vector length independence
- freedom from the need to do explicit-mode vector manipulation
- ease in specifying data allocation.

NSS I/O

The primary I/O activities of the machine are the input of initial problem parameters, restart from stored data, and the output of snapshots and restart dumps. Another important activity is the output of debug dumps. Two basic types of Navier-Stokes solutions are desired—steady and unsteady (or more correctly quasi-steady). Steady cases are characterized by the appearance of a solution that does not vary with time after some large number of time steps or large number of characteristic body lengths travelled. Unsteady cases are characterized by the appearance of a solution that is periodic in time after some large number of time steps. In order to analyze the unsteady or periodic nature of these solutions more time steps (on the order of six times that of steady cases) are required. Additional data output is also required in these cases. It is estimated that 75% of the time will be used to solve the steady flow case and the remaining 25% the unsteady.

The following output capabilities for these cases are desired.

- Snap Shots
  a. Integrated quantities such as drag, lift and moments approximately every 15-30 seconds.
  b. Surface quantities such as pressure and skin friction. If the grid moves with time, the grid co-ordinates must also be output. A given quantity such as pressure, plus the coordinates could total up to approximately 60,000 words of output every 15-30 seconds.
  c. Flow quantities in the field such as pressure or Mach number. For a grid of 1,000,000 points an entire field of, say, Mach numbers plus coordinates would be 4,000,000 words. However, it is anticipated that only selected grid points need to be output and this would be about one hundreth of the about to 40,000 words every 30 seconds. These snapshots require the heaviest output and for 60 minute runs would accumulate up to 5,000,000 words for the unsteady cases.
- Restart Dumps
- Debug Dumps
- Formatted I/O

FLOW MODEL CODE CHARACTERIZATION AND ANALYSIS

Codes supplied by NASA Ames were analyzed statically and dynamically to determine what the specific characteristics of the Flow Model problems are and how do they impact computer architecture. The codes studied were written for two specific computers. Features in each code that were specific to its target machine were stripped away to find the basic issues. The areas that were examined group themselves naturally into those issues which address processor requirements, memory requirements, or communications requirements, and are outlined below.

Memory Requirements

- Data Base Size - (The actual input/output variables)
- Program Size
- Workspace Size (Those variables never outputted in normal production code - the temporaries)
- Access Patterns (dimensionality of problems, subarray structure, indexing patterns)

Communications between Processors & Memories

- Number of Computations per Data Base Access
- Interaction of Problem Variables
- Data Dependency
- Control Structures
- Access Patterns (planes, rows, columns, etc.)

Processor Requirements

- Word Size and Format
- Relative frequency of operations
- Index computations
- Number of input operands per output operands
- Scalar operations
- Frequency of intrinsics
- Program structure

Each of these issues were examined in detail and the results are listed in Chapter 8 of the final report with a full discussion of the methodology.

The study of the memory requirements showed that the canonical problem variables and number of grid points produce a data base memory of 14-40 million words (NASA-Ames requirement). The workspace size was found to be approximately 40 temporaries per database variable. This of course is programmer and architecture dependent and hence is only an indication of the relationship between work space and data base. It was found that the problem arrays are generally 4 dimensional with 3 geometric and 1 variable coordinator. They are accessed in a fairly regular manner in the sense that the indexing is a function of the loop variables plus or minus a small integer. There is almost no indexing that occurs as a function of loop variable and another integer variable set outside of the loop. The structure of the loops indicate that entire arrays are processed in a given piece of the computation rather than small subarrays. Program size is relatively small at under 4000 card images.

Requirements on communication between processor and memory structure were determined by a number of flow-model...program parameters. The data dependency studies of variables in loops showed that there existed complex first order linear recurrences which were functions of each of the three geometric variables. These recurrences occurred in over 60% of the executing Implicit program. The study of the control or branching structures within the programs showed them to be relatively simple and generally linked to loop variables. Some were data dependent but when they occurred the variables were functions of inner loop parameters.

Further studies of the relationship between the data base memory requirements, the work space requirements and the number of floating point operations showed that a fetch or store to data base memory occurred infrequently in comparison to the number of floating point operations. Typically the Implicit (Steger) program has an average incidence of 15 floating point operations per fetch.

Additionally, by investigation of the indexing patterns within loop structures one found that there is relatively low interaction among problem variables on different grid points. For example, variables are fetched from several adjacent points, computations are performed and then a result is stored relative to the grid point. There is no continual switching back and forth of index patterns. The access patterns appear to be simple rows, columns and planes with a skip distance of 1.

Processer requirement studies showed that multiply, add, and multiply-add instructions are extremely important floating point operations. For example, in the Implicit Code it was found that 53% of all operations were multiplies, 44% were adds and 2.5% were divides. About 60% of all operations occurred as multiply-add pairs. Division and intrinsics as SQRT and EXP occur rarely and double precision is never required. Since most of the array references are to 3- and 4- dimensional arrays integer arithmetic calcula-

tions are a strong requirement. The combination of work space requirements and the average number of input operands to output operands (3.5) places certain requirements on the processor. NASA-Ames has additionally specified 10 digit accuracy requirement.

The data collected from the studies was used to define and delimit the characteristics of the requisite architecture. The output from the matching study together with the technology study and facilities study data were then used to develop definitions of an architecture discussed after the results of the facility study.

FACILITY STUDY

The primary objectives of this sub-study were threefold:

- Identify housing and support requirements of the facility
- To provide cost and schedule engineering estimates for effective planning
- Assessment of NSS implementation issues as they would impact architecture and design choices.

These objectives were pursued by determining the facility requirements of those units or sub-systems already identified and placing reasonable bounds on facility requirements for those elements which have yet to be specified. After a preliminary definition of the NSS, an implementation schedule and an engineering cost estimate were assembled, and analyzed. As the NSS definition proceeded, additional iterations on the schedule and cost were performed.

Finally, the critical issues relevant to implementing the NSS were defined and guidelines developed to insure that the design would indeed be realizable. This effort raised some interesting considerations which impacted the architecture choice and some design details as well.

Critical issues affecting the implementation or realization of the NSS in particular are:

CRITICAL PATH ANALYSIS was examined to eliminate short waterfalls in the schedule by locating their source and minimizing their occurrence.

PROCUREMENT problems can be avoided if there is an early identification of long-lead items, if custom componentry is minimized, if multiple sources are employed wherever possible, and if adequate protective documentation is obtained from each vendor. This issue can be the largest single risk factor in any program's schedule, cost, and possibly performance.

PRODUCTION considerations include maximizing the number of replicated units to minimize production learning curves and take advantage of economies of scale. Standardization of componentry, connectors, cables, etc., minimizes inventory problems and smoothes the production process.

MODULE OR SUBSYSTEM INTERFACE MANAGEMENT demands the reduction of complexity of interconnections between all functional elements.

DEBUGGING AND MAINTENANCE: As in the production considerations, if the number of complex elements, which field engineers must work with, are kept to a minimum, then debugging and maintenance are simplified -- furthermore, this minimizes the inventory of spares.

PACKAGING of any design must have the highest density consistent with heat removal. It must be such that the LRU (lowest replaceable unit) is easy to isolate, test and replace. Additionally, usage of common board types should be maximized.

LOGIC DESIGN RULES AND NOISE BUDGETS. A technology choice for the design must be mature enough to develop credible noise budgets, and provide adequate operational margins.

POWER. Finally, power consideration, suggest that we avoid complex power distribution schemes, and concurrently maximize the distribution of heat dissipation. These considerations will lead to some interesting features explained in the next section.

ARCHITECTURE EVOLUTION

The selection of the Synchronizable Array Machine for the NSS is presented as an evolution of concepts that grew out of the findings of the three sub-studies.

The first step in this evolution was the selection of a parallel architecture after examination of three generic types: hybrid, pipeline, and parallel. The hybrid was rejected for three reasons.

DIFFICULTY OF PROGRAMMING. Many difficulties make it impossible to translate the current Navier-Stokes algorithms to a hybrid machine. Years have already been spent in algorithm research in digital form. Even more investigation would be needed to recast the equations into suitable form for analog computation.

INACCURACIES, AND UNPREDICTABILITY OF THE INACCURACY. Such limited accuracy as exists in analog computation is often data dependent, and changes with age. In digital computation, any desired degree of accuracy can be specified.

COMPONENT FAILURES. Unlike a digital computation, where tests can continuously ensure that correct results are being produced, an analog computer has no error control. A faulty component or off-scale input produces an output voltage which is not distinguishable in kind from the output voltage of a properly functioning component.

Although analog processors have a very high computation rate, these limitations are totally unacceptable for the objectives of an NASF project.

Pipeline architectures as we know them today appear to suffer from inefficiencies, namely:

- Long start up times between vector operations,
- Difficulty in dealing with transpositions, and
- The need for massive amounts of work space memory to accommodate propagation of temporary variables.

Certainly these problems can be dealt with and solutions developed to make a pipeline a suitable architecture (as we have done for the parallel architecture) but a reexamination of the Facilities Study highlighted other issues which made the selection of a parallel array more sensible for Burroughs.

Assuming both architectures could be evolved to produce a design of equal performance, Burroughs is more confident that the parallel machine can be manufactured with less risk. The claim is based on observations:

- The large number of replicated units in a parallel array minimizes production and debugging and field engineering learning curves. Certain economics of scale could be realized in development as well.

- Burroughs experience, in three generations of parallel high performance systems (namely ILLIAC, PEPE, and the Burroughs Scientific Processor (BSP)), provides an invaluable data base of knowledge in the detailed design and manufacture of such a system.

The beginning of the architectural development, therefore, was based on the generic parallel configuration shown in Figure 4.

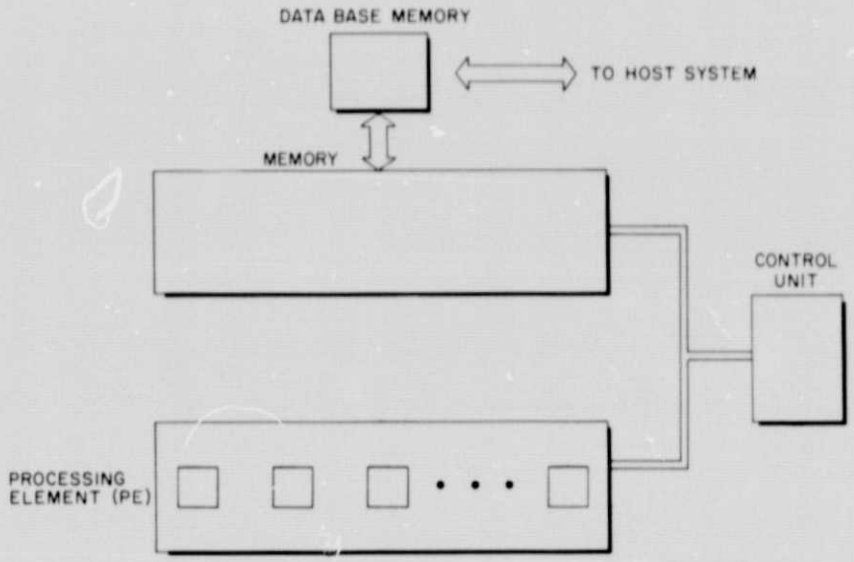


Figure 4 Parallel Configuration

From this point, the definition of SAM can be well understood as a series of refinements based on results of the sub-studies.

The ADI method of solution of the aerodynamic equations, with split operators, demands that many data arrays be transposed during access. The access patterns of this method require that 2-dimensional planes of the 3-dimensional grid be accessed in parallel. Planes are required from any 2 of 3 dimensions in the same grid. This implies the need for an efficient transposition mechanism.

Several different designs were considered. The selected Transposition Network (TN) is a unique innovation offering:

- low parts count
- minimal data access delay
- simple control requirements
- simple but flexible data allocation.

This design demands that memory be partitioned into a prime number of banks larger than the number of processors.

The Transposition Network (TN) is shown in Figure 5 as the first refinement of the generic parallel configuration.

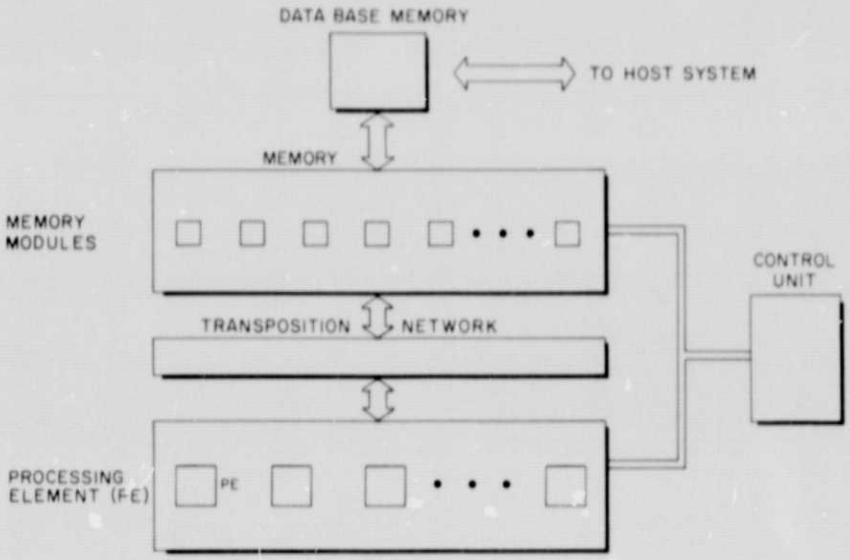


Figure 5 Parallel Configuration - Refinement 1

The occurrence of a significant number of floating point operations to fetches (especially in the implicit code) implies a large workspace requirement. In fact up to 40 temporary variables per data base variable may be generated. Propagation of such a large number of temporaries throughout the machine would cause severe timing penalties. To mitigate this problem, local memories for each processor are required. In addition, the bandwidth of the TN can then be reduced without performance degradation. This makes the Transposition Network simpler and less costly. The absence of data dependencies among points in the same plane allows this refinement (Figure 6) to occur. The increased cost of many data memories in the processor is offset by the decreased requirement for storage capacity in Extended Memory for temporary variables. The nomenclature for the main memory can now be appreciated as Extended Memory (EM).

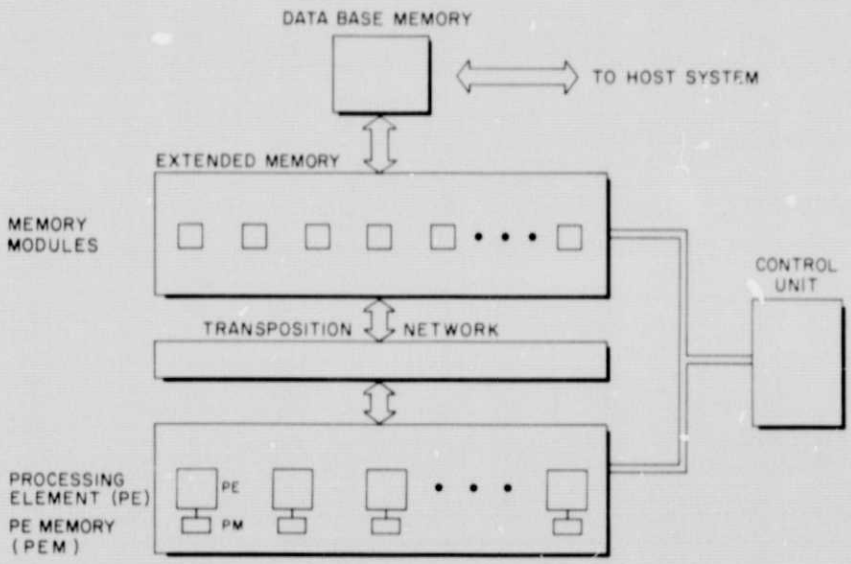


Figure 6 Parallel Configuration - Refinement 2

The result of this refinement allows one to think about parallelism as a series of vertical slices. That is:

Given a series of statements of the following form:

```
DOPARALLEL (one or more indices, say I, J, K, between limits)
  STATEMENT  1 -- involving variables indexed on the parallel indices
  STATEMENT  2 -- involving variables indexed on the parallel indices
       :
  STATEMENT  n -- involving variables indexed on the parallel indices
ENDDO
```

there are two ways of thinking of the parallelism.

In the first method, statement 1 is executed on the vectors implied by the parallel indices. Then statement 2 is executed as a vector statement, and so on up to the nth statement. Having each statement executed separately as a vector statement is called "horizontal slicing" of the parallelism.

The second method is to assign a processor to a particular instance of the set of indices. Processor 17, for example, may handle all computation associated with J=1 and K=19, while processor no. 222 handles J=3 and K=22. Each processor now executes, essentially independently, a piece of code involving the I index. This kind of parallelism has been called "vertical slicing." Vertical slicing is appropriate when, as in the Navier-Stokes equations, there is little interaction between the variables at one grid point and the variables at another.

Three or more generations of parallel processors have shown that instruction interpretation of parallel constructs by the CU creates a bottleneck. The CU must be extremely fast to keep up with the array. Its complexity is severe enough without this responsibility. The program size has been observed to be small enough to consider placing program memories in each processor as shown in Figure 7. This now results in a stand alone processor with manageable interface (very few lines) to the control unit, elimination of massive cabling and a simpler CU. These savings and their attendant design and schedule issues will offset the cost of multiple copies of the program memory, as well as improve performance.

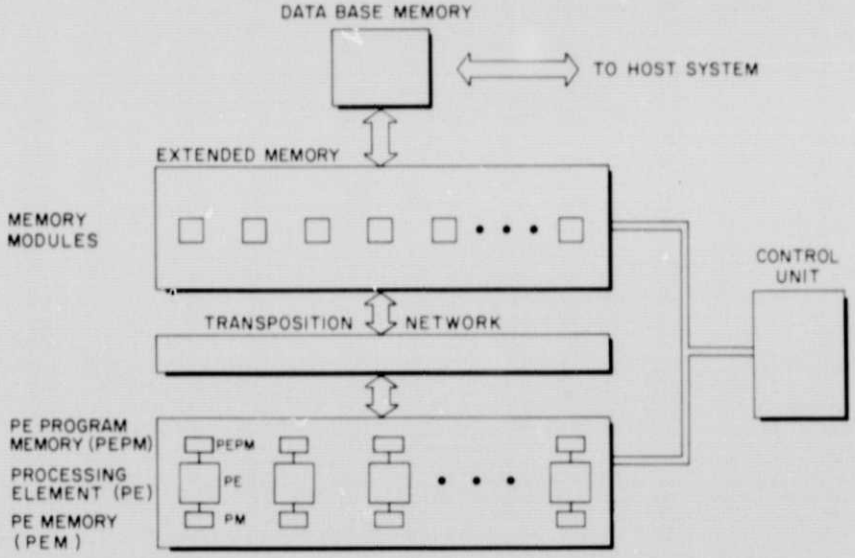


Figure 7 Parallel Configuration - Refinement 3

In a parallel array with a single program memory, the distribution of instructions by the CU serves to synchronize the operation of the PE's. Distribution of the program to local program memories results in a requirement for a synchronization mechanism between CU and PE's. To provide maximum flexibility, we elected to invoke the synch mechanism explicitly in the code stream (Figure 8). This allows synchronization to occur only when necessary, (i.e., just prior to parallel fetches and stores). Processors can run concurrently without waiting for each other, which permits data dependent instruction options (e.g. round after normalize if overflow) to be executed only when needed. The independence allows idle processors to execute confidence checks on themselves. Different code sequences for different areas of the airspace may be executed in different processors.

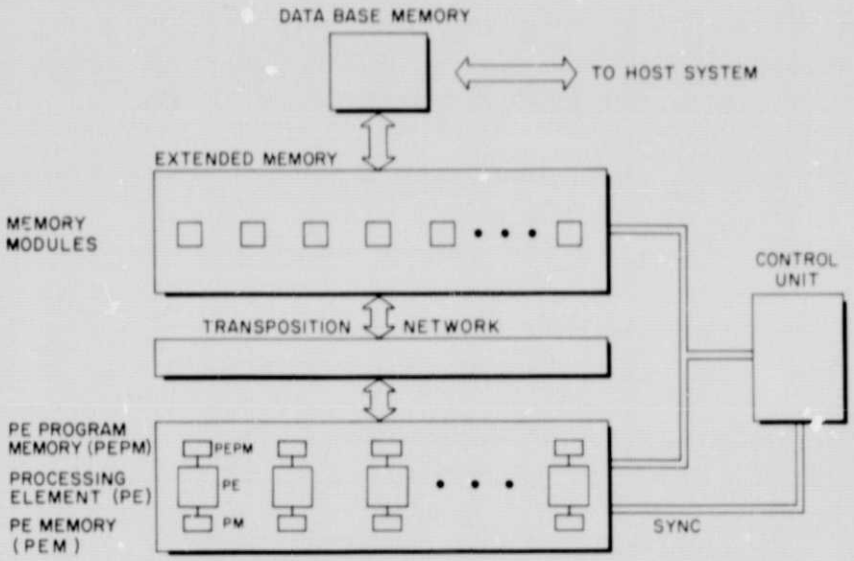


Figure 8 Parallel Configuration - Refinement 4

The choice of 512 as the number of processors is based primarily on the highest expected speed of efficient memory chips. 16k-bit static RAM chips are expected to be available at about 100 ns cycle time, by 1980, and are appropriate to processor and control unit memories. 64k-bit dynamic RAM chips are expected to be available at about the same time, at speeds nearly matching the present 200 ns or so speed of current 16k dynamic RAMs. These are the memory chips in the baseline system.

Consider, for example, the effect on the design of a choice of 256 processors. The twice-as-fast processor memories would require 50 ns chips, which would be available only in a 4k-bit size. Thus, the total number of memory chips would double, from the 37,888 memory chips of the baseline system to a total of 75,776 chips. The twice as fast EM would require 16k-bit chips to maintain the same speed, and its size would quadruple from 29,176 memory chips to 116,704 chips. Parts count in the twice-as-fast processor is estimated to double, making no net savings, but increasing the required design effort.

The size of data base for codes expected to execute for 10 minutes indicates as much as 10^{17} bits of data are operated upon. To expect no failures in that time is ambitious indeed, therefore it was necessary to impose a strict philosophy of fault detection and correction in the design of the hardware and software, including:

- Hardware Error Detection
- Hardware Error Correction
- Arithmetic Checking

Figure 9 is a block diagram of SAM, the Baseline Design for the NSS. Its evolution, as well as subsequent design decisions and guidelines results in a design which features the items described below.

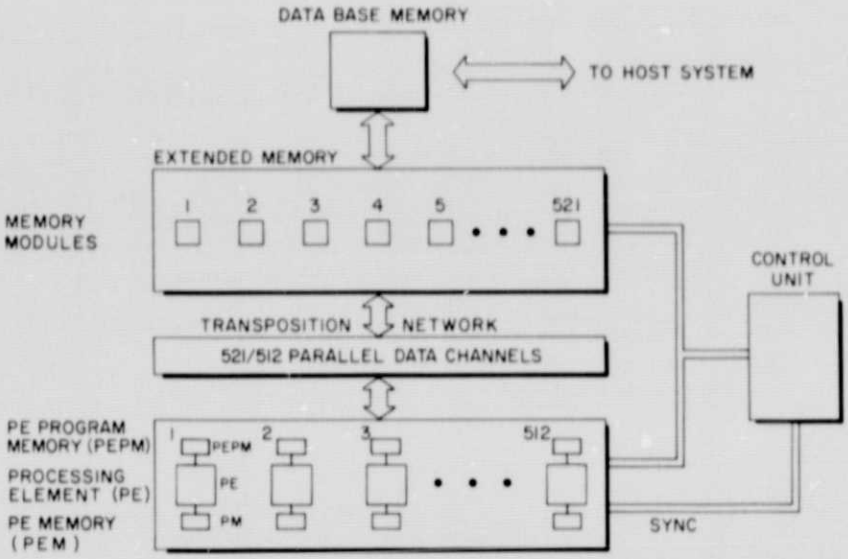


Figure 9 Parallel Configuration - Refinement 5

HIGH THROUGHPUT

The throughput potential of the NSS is 1.7 billion Floating Point Operations per second. This is derived from the selective ratios of the instruction mix combined with the expected execution time of the operations. This yields 294 nsec per 512 floating point operations which is equivalent to 1.7 billion FLOPS. Additional study of the baseline for the specific codes indicates that the required effective rate of 1 billion FLOPS is achievable.

EASE OF USE

High level language requirements, the guidelines of matching machine code to the user language and indeed the use of a High Level Language to write the compiler were inportant decisions made early in the study. The Vertical Slice Concept allows all classical serial optimization techniques to be utilized on the SAM. Recognizing that this architecture has unprecedented flexibility, it is incumbent upon the compiler to have debug aids to protect the user.

The protected environment in which SAM operates – the high speed computational envelope isolated from the rest of the system – requires that it have only a very small operating system of its own. I/O to and from that envelope will not encumber the user or SAM as well. A typical work flow is illustrated in Figure 10.

This architecture is a tradeoff optimized for the aerodynamic problem, yielding lesser performance for:

- Problems with intimate arithmetic data dependency from one grid point variable to another,
- Interactive environments, and
- Multi-programming environments.

We feel this represents a unique solution to the problem of numerical aerodynamic simulation, and Burroughs presents this design with full confidence in its feasibility. We believe that this system is the best approach to meeting the NASF goals in a timely and cost-effective manner, maintaining NASA's position in the forefront of scientific endeavor.

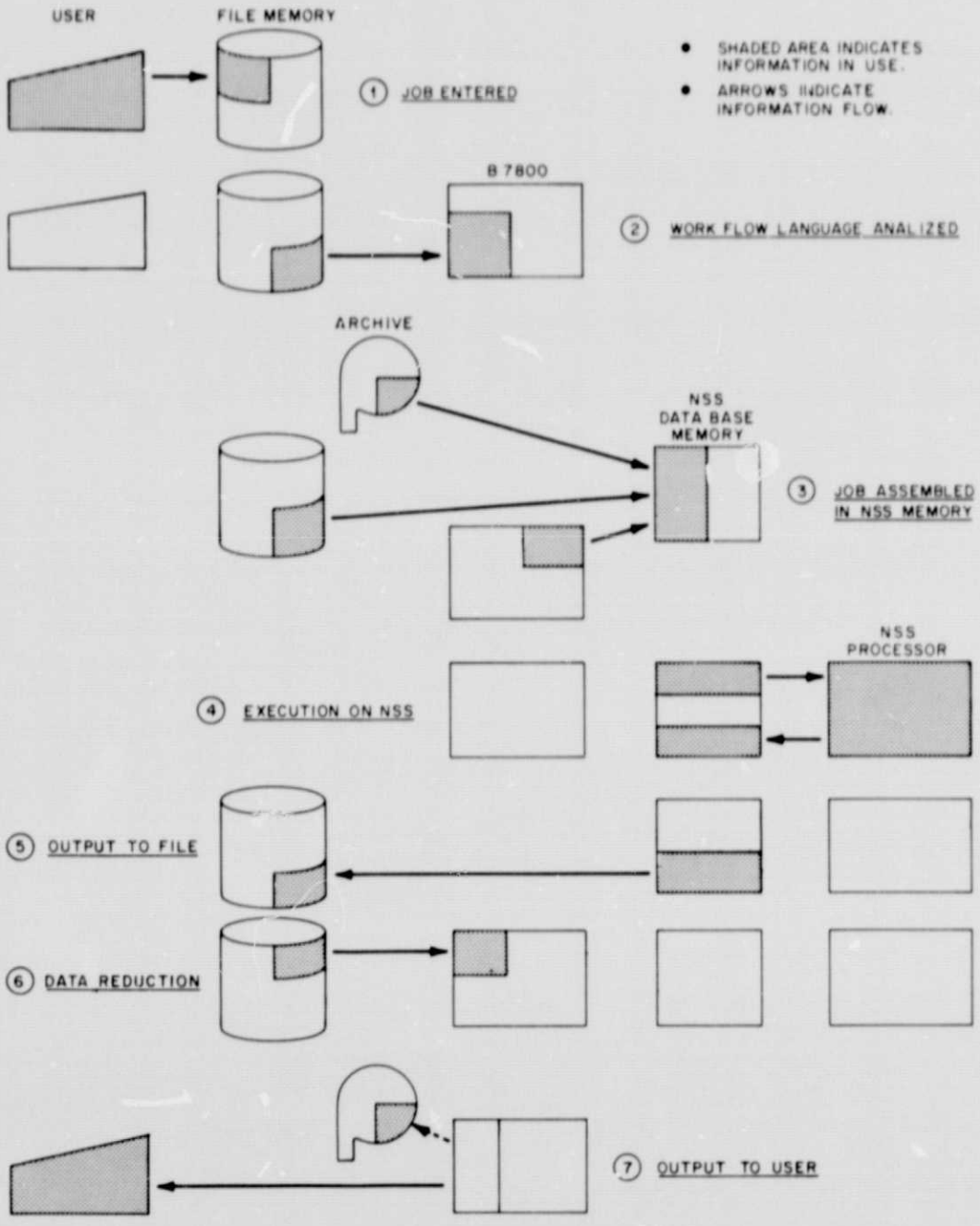


Figure 10 Typical Work - Flow Schematic